Client Ref. No. P17385
Attorney Docket No. INTCP005

APPLICATION FOR UNITED STATES PATENT

SYSTEMS AND METHODS USING NIC-BASED PREFETCHING FOR HOST TCP CONTEXT LOOKUP

INVENTOR: **David B. Minturn**
17125 Chehalem Way
Hillsboro, OR 97123
A Citizen of United States

ASSIGNEE: **Intel Corporation**
2200 Mission College Blvd.
Santa Clara, CA 95052
A DELAWARE CORPORATION

ENTITY: **Large**

Jung-hua Kuo
Attorney at Law
P.O. Box 3275
Los Altos, CA 94024
Tel: (650) 988-8070
Fax: (650) 988-8090

SYSTEMS AND METHODS USING NIC-BASED PREFETCHING FOR HOST TCP CONTEXT LOOKUP

BACKGROUND OF THE INVENTION

[0001] A network generally refers to computers and/or other device interconnected for data communication. A network interface controller (NIC) is a hardware device that connects a host computer system to a computer network such as a local area network (LAN). The NIC communicates with the host bus and is controlled by the host CPU in a manner similar to the way the host CPU controls an I/O device. Thus, the NIC appears as an I/O device to the host computer. To the network, the NIC can send and receive packets and appears as an attached computer.

[0002] NICs typically use descriptor rings for processing packets both in the receive direction and in the transmit direction. For example, when the NIC receives a packet or frame, the NIC reads a receive descriptor from the host system memory to determine where to place the data in the host system. After the data is moved to the host system main memory, the receive descriptor is written back out to the host system memory with status information about the received frame. In the transmit direction, the NIC operates in a similar fashion to the receive direction. In particular, the NIC is first notified by the host CPU of a new transmit. The NIC then reads the descriptor to locate the data, reads the data, and then writes the descriptor back with status about the transmit. On transmits, the NIC typically reads the next expected descriptor to see if any more data needs to be sent. As is evident, each receive or transmit frame results in at least three peripheral component interconnect (PCI) or peripheral bus reads or writes in addition to a status register read.

[0003] After the NIC receives and transfers the packet to the host computer, the host computer processes the packet through a protocol stack. During the host packet processing, the protocol header (e.g., transmission control protocol (TCP), Internet protocol (IP), Ethernet) is removed. The data portion is thus recovered and can be made available to a user, an application program, etc.

[0004] A non-insignificant amount of processing time is required for the host system to identify a TCP context due in part to memory access times. While the TCP context lookup may be offloaded to the NIC, such a mechanism requires significantly more NIC memory as well as elaborate communications between the NIC and the host to manage "statefull" memory on the NIC.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements.

[0006] **FIG. 1** is a block diagram of an illustrative host computer system with a network interface controller (NIC) for interfacing with a network.

[0007] **FIG. 2** is a block diagram illustrating the NIC in more detail.

[0008] **FIG. 3** illustrates host TCP data structures.

[0009] **FIG. 4** illustrates NIC TCP data structures.

[0010] **FIG. 5** is a flowchart illustrating a process for processing packets received by the NIC.

[0011] **FIG. 6** is a flowchart illustrating an alternative process in which the host, rather than the NIC, performs the prefetch operations.

DESCRIPTION OF SPECIFIC EMBODIMENTS

[0012] Systems and methods using network interface card-based (NIC-based) prefetching for host TCP context lookup are disclosed. The following description is presented to enable any person skilled in the art to make and use the invention. Descriptions of specific embodiments and applications are provided only as examples and various modifications will be readily apparent to those skilled in the art. The general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is to be accorded the widest scope encompassing numerous alternatives, modifications and equivalents consistent with the principles and features disclosed herein. For purpose of clarity, details relating to technical material that is known in the technical fields related to the invention have not been described in detail so as not to unnecessarily obscure the present invention. It should be appreciated that the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, a device, a method, or a computer readable medium such as a computer readable storage medium or a computer network wherein program instructions are sent over optical or electronic communication lines. Several inventive embodiments of the present invention are described below.

[0013] The process generally includes hashing, by the NIC, a packet received over the network, computing a host hash table cache line in a host memory using the hash value and using a hash table pages table containing host memory physical page addresses

of a host hash table, and computing a host context table cache line in a host memory using the hash value and using a context table pages table containing host memory physical page addresses of a host context table. The NIC may be initialized with the hash table pages table and the context table pages table as well as with the a set number of hash node entries in the hash table of the host memory.

[0014] The NIC generally includes a hashing logic to generate a hashing value from a packet received over a network, a hash table pages table and a context table pages table for storing host memory physical page addresses of a host hash table and a host context table, respectively, and a cache line determinator configured to associate the hash value with a host hash table cache line and a host context table cache line in a host memory using the hash table pages table and the context table pages table, respectively.

[0015] A computer program product may also be implemented. The computer program product disposed on a computer readable medium to process packets, the program generally includes instructions for causing at least one processor to hash, by a NIC in communication with a host and the network, a packet received from the network, the packet having a context associated therewith to generate a hash value from context of the received packet, compute a host hash table cache line in a host memory using the hash value and using a hash table pages table containing host memory physical page addresses of a host hash table, and compute a host context table cache line in a host memory using the hash value and using a context table pages table containing host memory physical page addresses of a host context table.

[0016] These and other features and advantages of the present invention will be presented in more detail in the following detailed description and the accompanying figures which illustrate by way of example the principles of the invention.

[0017] **FIG. 1** is a block diagram of an illustrative host computer system 100 with a network interface card (NIC) 102 for interfacing with a network 110 such as a local area network (LAN). The host computer system 100 also includes a CPU 104, memory 106, and other I/O devices 112 in addition to the NIC 102. The various components of the host computer system 100 communicate with each other over a host bus 108. The host computer system 100 generally refers to the computer system that processes the network protocol stack, e.g., transmission control protocol (TCP), Internet protocol (IP), Ethernet. The NIC 102 includes hash logic for performing hashing functions. A hash table or "hashing" is the creation of an index to the table content derived from a transformation of the information in the packet header A hash table or index of the stored information facilitates efficient searching of the information being stored.

[0018] **FIG. 2** is a block diagram illustrating the NIC 102 in more detail. As shown, the NIC 102 receives TCP/IP packets from the network 110. Although not shown, the NIC 102 also forwards TCP/IP packets to the network 110. The NIC includes a hashing logic 114 for performing hashing functions and a cache line determinator 116 for computing starting cache line locations in the host memory using the hash value as will be described in more detail below. The NIC 102 also includes a NIC resident hash table pages table 142 and a NIC resident TCB context table pages table 144. The hash table pages table 142 and the TCB context table pages table 144 contain the physical page addresses of the host TCP hash table and host TCB context table, respectively. The tables 142, 144 may be pre-loaded into the NIC 102 by the host system upon system initialization.

[0019] The host and NIC TCP data structures are illustrated in **FIGS. 3** and **4**, respectively. As shown in **FIG. 3**, the host resident TCP data structures include a host

hash node table 120 and a host TCP-controlled block (TCB) context table 130. The host hash node table 120 is a data structure that contains multiple host TCP hash node entries 122.

[0020] The host hash node table 120 and the host TCB context table 130 may be located in contiguous virtual memory. The TCP host hash node table 120 and the host TCB context table 130 should be pre-pinned, i.e., pre-allocated in a known physical address space. These host tables can be configured to use any suitable memory page size such as 4K or 4M size pages. A larger page size reduces the NIC memory requirements. The NIC is also initialized with the size of the host TCB context table entries and the host TCP hash table entries and page size.

[0021] The host hash node table 120 contains a set number (N) TCP hash node entries 122 and a set number of contiguous cache lines. Each host TCP hash node entry 122 is a data structure containing one or more TCP context references 124, 126, i.e., TCP context identifier information values. Each TCP context reference 124, 126 includes source and destination IP addresses as well as the source and destination TCP ports. In the example shown in **FIG. 3**, each host TCP hash node entry 122 may contain up to X number of entries. Each host TCP hash node entry 122 can be one or more consecutive cache lines in size with each cache line containing multiple TCP context references 124, 126.

[0022] In sum, in the example shown in **FIG. 3**, there are N hash nodes, X entries per hash node, and Z entries in TCB context table. Merely as an example, there may be N = 1600 hash nodes and X = 8 entries per hash node for a total of up to Z = 12,800 entries in TCB context table.

[0023] The other host resident data structure is the host TCB context table 130. The host TCB context table 130 is preferably a virtually contiguous data structure that contains the TCP context information, each entry corresponding to one of the TCB context references 124, 126. The host TCB context table 130 is preferably divided between a main host TCB context table 132 and a secondary host TCB context table 134. The main host TCB context table 132 preferably includes N entries corresponding to the first TCB context reference entries (entry[0]) 124 of the N node entries 122. The secondary host TCB context table 134 contains Z-N entries corresponding to all the remaining TCB context references 126 of the hash node entries 122.

[0024] For the NIC data structures shown in **FIG. 4**, the NIC TCP data structures 140 include a NIC resident hash table pages table 142 and a NIC resident TCB context table pages table 144. The NIC hash table pages table 142 is a table in NIC memory containing the physical address of each page of the host TCP hash node table (reference number 120 in **FIG. 3**). The NIC TCB context table pages table 144 is a table in NIC memory containing the physical address of each page of the host TCB context table (reference number 130 in **FIG. 3**). The NIC hash table pages table 142 and the NIC TCB context table pages table 144 have the same number of entries and each entry may be, for example, 64 bits wide.

[0025] The mechanism described herein has minimal memory requirements on the NIC. The host tables can be configured to use any suitable memory page size such as 4K or 4M size pages. As noted above, a larger page size reduces the NIC memory requirements. For example, with a 4K byte page size, 512 byte context size, and a 64 bit address, approximately 8 bits of NIC memory space is needed per TCB entry. This is derived from (64 bit address/page) * (512 byte context size) / (4K byte/page). In

contrast, with a 4M page size, using the same derivation, only 0.008 bits of NIC memory space is needed per TCB entry.

[0026] Similarly, with a 4K byte page size, 64 byte hash nodes, and a 64 bit address, approximately 1 bit of NIC memory space is needed per hash node. This is derived from (64 bit address/page) * (64 byte hash node size) / (4K byte/page). In contrast, with a 4M page size, using the same derivation, 0.001 bits of NIC memory space is needed per TCB entry.

[0027] The general data structures of the host and the NIC having been presented, the process 150 for processing packets received by the NIC will now be described with reference to **FIG. 5**. It is noted that although not shown in **FIG. 5**, prior to processing any packets, the host system is initialized along with the NIC. Upon system initialization, the host resident software pre-loads the NIC TCP hash table pages table and the NIC TCB context table pages table containing the physical page addresses of the host TCP hash table and host TCB context table, respectively. These tables can be configured to use any suitable memory page size such as 4K or 4M size pages. A larger page size reduces the NIC memory requirements. The NIC is also initialized with the size of the host TCB context table entries and the host TCP hash table entries.

[0028] The process 150 for processing packets received by the NIC begins when the NIC receives an incoming TCP/IP packet 152. The NIC performs a hash of the received packet tuples 154, i.e., the TCP connection identification information, to obtain a logical hash table index. The TCP context information includes the IP source and destination addresses and the TCP source and destination ports. The hash table index serves as an index to the NIC hash table pages table and to the TCB context table pages table.

[0029] The NIC then calculates the starting cache line locations in both the host hash node table and the host TCB context table by using calculated hash table index, i.e., the hash value, as an index into the NIC hash table pages table and the NIC TCB context table pages table. These starting cache line locations are referred to as host hash node table prefetch and the host TCB context table prefetch.

[0030] Specifically, the NIC computes the hash node page and the starting hash node cache line location 156 using the logical table index derived in 154. Note that the physical address of the hash node page is obtained from the NIC hash table pages table (142 in **FIG. 4**). For example, the hash node page and the starting hash node cache line location (HN_CL) may be determined by:

hash node page = (index * size of hash node entry) / (page size); and

HN_CL = physical address of the hash node page +

offset of the logical table index within the hash node page

[0031] Similarly, the NIC computes the TCB context table page and the starting TCB context cache line location 158 using the logical table index derived in 154. Note that the physical address of the hash node page can be obtained from the NIC hash table pages table (142 in **FIG. 4**). For example, the TCB context table page and the starting TCB context cache line location (TCB_CL) may be determined by:

TCB table page = (index * size of TCB context entry) / (page size); and

TCB_CL = physical address of the TCB context page +

offset of the logical table index within the TCB context page.

[0032] The NIC then issues an I/O prefetch bus operation to fetch hash node cache lines and TCB context cache lines 160. In particular, the starting cache line locations HN_CL in the host hash node table and TCB_CL in the host TCB context table are used

for the host hash node table and TCB context prefetch operations. The HN_CL and TCB_CL give the host caching subsystem a hint as to where the target TCB context is stored. As noted above, only the first entry of each node in the host hash node table is stored in the main host TCB context table while the remainder of the entries are stored in the secondary host TCB context table. Thus, in a best case scenario, the target TCB context is stored in the main host TCB context table such that no further searching is needed in order to located the target TCB context. If the target TCB context is stored in the secondary host TCB context table, then some amount of additional searching would be performed.

[0033] After the I/O prefetch is issued, the NIC writes the packet header and payload into the host memory 162 along with the receive descriptor 164.

[0034] **FIG. 6** is a flowchart illustrating an alternative process 150A in which the host, rather than the NIC, performs the prefetch operations. After the NIC determines the starting cache line locations in both the host hash node table and the host TCB context table 156, 158, the NIC may add the starting cache line locations in both the host hash node table (i.e., the host hash node table prefetch) and the host TCB context table (i.e., the host TCB context table prefetch) to the NIC receive descriptor that is associated with the packet received 160A. Then, after the NIC writes the packet header, packet payload, and receive descriptor to host memory 162, 164, the host resident packet driver issues software prefetch instructions using the host hash node table prefetch and the host TCB context table prefetch 160B. In particular, the host resident packet driver reads the NIC packet descriptor from the host memory and subsequently issue the processor prefetch commands using the host hash node table prefetch and the host TCB context prefetch as the cache line identifiers. In this embodiment, the cache line addresses passed by the

NIC to the host NIC driver may be virtual addresses such that the NIC does not need to contain the two physical address translation tables, thereby simplifying the NIC requirements.

[0035] The above-described mechanism using NIC-based pre-fetching for host TCP context lookup provides several advantages. The mechanism facilitates in reducing the amount of processing time required by the host system to identify a TCP context as a result of reduced memory access times of accessing both the host TCP hash table and host TCP context. The mechanism also requires only one lookup for each TCP packet received. In addition, the mechanism is stateless in that once the NIC is initialized there is no additional communication required between the NIC and the host for purposes of the TCP context lookup. The mechanism has minimal memory requirements on the NIC. Thus, host TCP efficiency is increased as a result of reducing or minimizing the impact of memory latency on host TCB entry lookup, i.e., classification. Thus, the combination of the data structure layout of the host and the NIC, the NIC processing and lookup, and a host CPU or NIC prefetch functionality assists the host-based TCP processing stack.

[0036] While various embodiments of the present invention are described and illustrated herein, it will be appreciated that they are merely illustrative and that modifications can be made to these embodiments without departing from the spirit and scope of the invention. Thus, the invention is intended to be defined only in terms of the following claims.